In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:

```
df = pd.read_csv('tweets.csv')
df
```

Out[ ]:

	id	label	tweet
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1...
1	2	0	Finally a transparant silicon case ^^ Thanks t...
2	3	0	We love this! Would you go? #talk #makememorie...
3	4	0	I'm wired I know I'm George I was made that wa...
4	5	1	What amazing service! Apple won't even talk to...
...	...	...	...
7915	7916	0	Live out loud #lol #liveoutloud #selfie #smile...
7916	7917	0	We would like to wish you an amazing day! Make...
7917	7918	0	Helping my lovely 90 year old neighbor with he...
7918	7919	0	Finally got my #smart #pocket #wifi stay conne...
7919	7920	0	Apple Barcelona!!! #Apple #Store #BCN #Barcelo...

7920 rows × 3 columns

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7920 entries, 0 to 7919
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      7920 non-null   int64 
 1   label   7920 non-null   int64 
 2   tweet   7920 non-null   object
dtypes: int64(2), object(1)
memory usage: 185.8+ KB
```

In [ ]:

```
df['label'].value_counts()
```

Out[ ]:

```
0    5894
1    2026
Name: label, dtype: int64
```

In [ ]:

```
df.isna().sum()
```

Out[ ]:

```
id       0
label    0
tweet    0
dtype: int64
```

Tweets preprocessing

In [ ]:

```
!pip install unidecode
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting unidecode
  Downloading Unidecode-1.3.6-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.9/235.9 kB 10.5 MB/s eta 0:00:00
Installing collected packages: unidecode
Successfully installed unidecode-1.3.6
```

In [ ]:

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
import re
import unidecode
from nltk.tokenize.toktok import ToktokTokenizer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [ ]:

```
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"I'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\^^", "", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)
    return text
```

In [ ]:

```
def clean_keywords(word):
    return re.sub(r'%20', ' ', word)
def to_lowercase(word):
    return word.lower()
def remove_accents(word):
    return unidecode.unidecode(word)
def remove_punctuation(word):
    return re.sub(r"[!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n -' ]"," ",word)
```

In [ ]:

```
def cleaning_URLs(word):
    return re.sub('((www.[^s]+)|(https?:\/\/.*?[\s+]))',' ',word)
def remove_mentions(word):
    return re.sub('@[\w]*',' ',word)
```

In [ ]:

```
#Setting English stopwords
tokenizer1 = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer1.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
```

Removing all hyperlinks

In [ ]:

```
df['cleaned_tweet'] = df['tweet'].apply(lambda x: cleaning_URLs(x))
```

Removing and replacing certain patterns

In [ ]:

```
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: remove_special_characters(x))
```

Removing @mentions of users

In [ ]:

```
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: remove_mentions(x))
```

Removing all special characters

In [ ]:
```
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: remove_punctuation(x))
```

Converting everything to unicode characters

In [ ]:
```
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: remove_accents(x))
```

Convert everything to lowercase

In [ ]:
```
df['cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: to_lowercase(x))
```

Removing stopwords using NLTK corpus library

In [ ]:
```
df['final_cleaned_tweet'] = df['cleaned_tweet'].apply(lambda x: remove_stopwords(x, True))
```

In [ ]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

Applying stemming

In [ ]:
```
def simple_lemmatizer(text):
    text= ' '.join([lemmatizer.lemmatize(word) for word in text.split()])
    return text
```

In [ ]:
```
df['final_cleaned_tweet']= df['final_cleaned_tweet'].apply(simple_lemmatizer)
df['cleaned_tweet']= df['cleaned_tweet'].apply(simple_lemmatizer)
```

In [ ]:
```
df
```

Out[ ]:

	id	label	tweet	cleaned_tweet	final_cleaned_tweet
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1...	fingerprint pregnancy test android apps beauti...	fingerprint pregnancy test android apps beauti...
1	2	0	Finally a transparant silicon case ^^ Thanks t...	finally a transparant silicon case thanks to m...	finally transparant silicon case thanks uncle ...
2	3	0	We love this! Would you go? #talk #makememorie...	we love this would you go talk makememories un...	love would go talk makememories unplug relax i...
3	4	0	I'm wired I know I'm George I was made that wa...	im wired i know im george i wa made that way i...	im wired know im george made way iphone cute d...
4	5	1	What amazing service! Apple won't even talk to...	what amazing service apple wont even talk to m...	amazing service apple wont even talk question ...
...	...	...	...	...	...
7915	7916	0	Live out loud #lol #liveoutloud #selfie #smile...	live out loud lol liveoutloud selfie smile son...	live loud lol liveoutloud selfie smile sony mu...
7916	7917	0	We would like to wish you an amazing day! Make...	we would like to wish you an amazing day make ...	would like wish amazing day make every minute ...
7917	7918	0	Helping my lovely 90 year old neighbor with he...	helping my lovely 90 year old neighbor with he...	helping lovely 90 year old neighbor ipad morni...
7918	7919	0	Finally got my #smart #pocket #wifi stay conne...	finally got my smart pocket wifi stay connecte...	finally got smart pocket wifi stay connected a...
7919	7920	0	Apple Barcelona!!! #Apple #Store #BCN #Barcelo...	apple barcelona apple store bcn barcelona trav...	apple barcelona apple store bcn barcelona trav...

7920 rows × 5 columns

Bag of Words model

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [ ]:
```
bow_model = CountVectorizer(stop_words="english", ngram_range=(1,1))
bow_vector = bow_model.fit_transform(df['final_cleaned_tweet']).todense()
```

In [ ]:
```
bow_df = pd.DataFrame(bow_vector)
bow_df.columns = sorted(bow_model.vocabulary_)
bow_df.head()
```

Out[ ]:

	000	00000	002	004	0051	007	008	01	010111	0101am	...	zooper	zsofimonster	ztjeq	zumies	zune	zunehd	zurich	zv
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 20392 columns

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(bow_df, df['label'], test_size=0.15, random_state=134)
```

Using Logistic Regression

In [ ]:

```
bow_log = LogisticRegression(fit_intercept=False)
bow_log.fit(x_train, y_train)
y_pred_bow_log = bow_log.predict(x_test)
print("Accuracy score of Bag of words model using logistic regression: " + str(round(accuracy_score(y_test, y_
```

```
Accuracy score of Bag of words model using logistic regression: 88.05%
```

Using Decision Tree Classifier

In [ ]:

```
bow_dt = DecisionTreeClassifier()
bow_dt.fit(x_train, y_train)
y_pred_bow_dt = bow_dt.predict(x_test)
print("Accuracy score of Bag of words model using Decision Tree Classifier: " + str(round(accuracy_score(y_test
```

```
Accuracy score of Bag of words model using Decision Tree Classifier: 84.85%
```

Using Gaussian Naive Bayes

In [ ]:

```
bow_gnb = GaussianNB()
bow_gnb.fit(x_train, y_train)
y_pred_bow_gnb = bow_gnb.predict(x_test)
print("Accuracy score of Bag of words model using Gaussian Naive Bayes: " + str(round(accuracy_score(y_test, y_
```

```
Accuracy score of Bag of words model using Gaussian Naive Bayes: 78.96%
```

TFIDF Model

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:

```
tfidf = TfidfVectorizer()
tdfif_dense = tfidf.fit_transform(df['final_cleaned_tweet']).todense()
```

In [ ]:

```
tfidf_df = pd.DataFrame(tdfif_dense)
tfidf_df
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	20550	20551	20552	20553	20554	20555	20556	20557	20558	20559
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7915	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7916	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7917	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7918	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7919	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

7920 rows × 20560 columns

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(tfidf_df, df['label'], test_size=0.15, random_state=134)
```

Using Logistic Regression

In [ ]:

```
tfidf_log = LogisticRegression(fit_intercept=False)
tfidf_log.fit(x_train, y_train)
y_pred_tfidf_log = tfidf_log.predict(x_test)
print("Accuracy score of TFIDF model using logistic regression: " + str(round(accuracy_score(y_test, y_pred_tf
```

```
Accuracy score of TFIDF model using logistic regression: 88.05%
```

Using Decision Tree Classifier

In [ ]:

```
tfidf_dt = DecisionTreeClassifier()
tfidf_dt.fit(x_train, y_train)
y_pred_tfidf_dt = tfidf_dt.predict(x_test)
print("Accuracy score of TFIDF model using Decision Tree Classifier: " + str(round(accuracy_score(y_test, y_pre
```

```
Accuracy score of TFIDF model using Decision Tree Classifier: 83.75%
```

Using Gaussian Naive Bayes

In [ ]:

```
tfidf_gnb = GaussianNB()
tfidf_gnb.fit(x_train, y_train)
y_pred_tfidf_gnb = tfidf_gnb.predict(x_test)
print("Accuracy score of TFIDF model using Gaussian Naive Bayes: " + str(round(accuracy_score(y_test, y_pred_t
```

```
Accuracy score of TFIDF model using Gaussian Naive Bayes: 79.04%
```

Word Embeddings Models

In [ ]:

```
from gensim.models import Word2Vec as wtv
```

In [ ]:

```
preprocessed_text = df['cleaned_tweet'].apply(lambda x: x.split())
preprocessed_text
```

Out[ ]:

```
0       [fingerprint, pregnancy, test, android, apps, ...
1       [finally, a, transparant, silicon, case, thank...
2       [we, love, this, would, you, go, talk, makemem...
3       [im, wired, i, know, im, george, i, wa, made, ...
4       [what, amazing, service, apple, wont, even, ta...
                              ...
7915    [live, out, loud, lol, liveoutloud, selfie, sm...
7916    [we, would, like, to, wish, you, an, amazing, ...
7917    [helping, my, lovely, 90, year, old, neighbor,...
7918    [finally, got, my, smart, pocket, wifi, stay, ...
7919    [apple, barcelona, apple, store, bcn, barcelon...
Name: cleaned_tweet, Length: 7920, dtype: object
```

Creating Cbow & skipgram models

In [ ]:

```
cbow_w2v_model = wtv(preprocessed_text, vector_size=800, window=5, min_count=3, sg=0)
skgram_w2v_model = wtv(preprocessed_text, vector_size=800, window=5, min_count=3, sg=1)
```

In [ ]:

```
print("cbow vocabulary size:", len(cbow_w2v_model.wv.index_to_key))
print("skipgram vocabulary size:", len(skgram_w2v_model.wv.index_to_key))
```

```
cbow vocabulary size: 3943
skipgram vocabulary size: 3943
```

Function to return average word embedding vector value

In [ ]:

```
def get_embedding_w2v(doc_tokens, model):
    embeddings = []
    for tok in doc_tokens:
      if tok in model.wv.index_to_key:
          embeddings.append(model.wv.get_vector(tok))
    return np.mean(embeddings, axis=0)
```

Skipgram model

In [ ]:

```
X_x2v_model = preprocessed_text.apply(lambda x: get_embedding_w2v(x, skgram_w2v_model))
X_df_sg = pd.DataFrame(X_x2v_model.to_list())
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(X_df_sg, df['label'], test_size=0.15, random_state=134)
```

Using Logistic Regression

In [ ]:

```
sg_log = LogisticRegression(fit_intercept=False)
sg_log.fit(x_train, y_train)
y_pred_sg_log = sg_log.predict(x_test)
print("Accuracy score of Skipgram model using logistic regression: " + str(round(accuracy_score(y_test, y_pred_
```

```
Accuracy score of Skipgram model using logistic regression: 88.3%
```

Using Decision tree classifier

In [ ]:

```
sg_dt = DecisionTreeClassifier()
sg_dt.fit(x_train, y_train)
y_pred_sg_dt = sg_dt.predict(x_test)
print("Accuracy score of Skipgram model using Decision Tree Classifier: " + str(round(accuracy_score(y_test, y_
```

```
Accuracy score of Skipgram model using Decision Tree Classifier: 83.67%
```

Using Gaussian Naive Bayes

In [ ]:

```
sg_gnb = GaussianNB()
sg_gnb.fit(x_train, y_train)
y_pred_sg_gnb = sg_gnb.predict(x_test)
print("Accuracy score of Skipgram model using Gaussian Naive Bayes: " + str(round(accuracy_score(y_test, y_pred
```

```
Accuracy score of Skipgram model using Gaussian Naive Bayes: 82.74%
```

Cbow model

In [ ]:

```
X_x2v_model = preprocessed_text.apply(lambda x: get_embedding_w2v(x, cbow_w2v_model))
X_df_cbow = pd.DataFrame(X_x2v_model.to_list())
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(X_df_cbow, df['label'], test_size=0.15, random_state=134)
```

In [ ]:

```
cbow_log = LogisticRegression(fit_intercept=False)
cbow_log.fit(x_train, y_train)
y_pred_cbow_log = cbow_log.predict(x_test)
print("Accuracy score of Cbow model using logistic regression: " + str(round(accuracy_score(y_test, y_pred_cbow
```

```
Accuracy score of Cbow model using logistic regression: 84.6%
```

In [ ]:

```
cbow_dt = DecisionTreeClassifier()
cbow_dt.fit(x_train, y_train)
y_pred_cbow_dt = cbow_dt.predict(x_test)
print("Accuracy score of Cbow model using Decision Tree Classifier: " + str(round(accuracy_score(y_test, y_pred
```

```
Accuracy score of Cbow model using Decision Tree Classifier: 80.47%
```

In [ ]:

```
cbow_gnb = GaussianNB()
cbow_gnb.fit(x_train, y_train)
y_pred_cbow_gnb = cbow_gnb.predict(x_test)
print("Accuracy score of Cbow model using Gaussian Naive Bayes: " + str(round(accuracy_score(y_test, y_pred_cbo
```

```
Accuracy score of Cbow model using Gaussian Naive Bayes: 77.27%
```

DeepLearning Model

In [ ]:

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [ ]:

```
tweets = df['final_cleaned_tweet'].to_list()
labels = df['label'].to_list()
```

In [ ]:

```
tokenizer = Tokenizer(oov_token='<oov>')
tokenizer.fit_on_texts(tweets)
```

In [ ]:

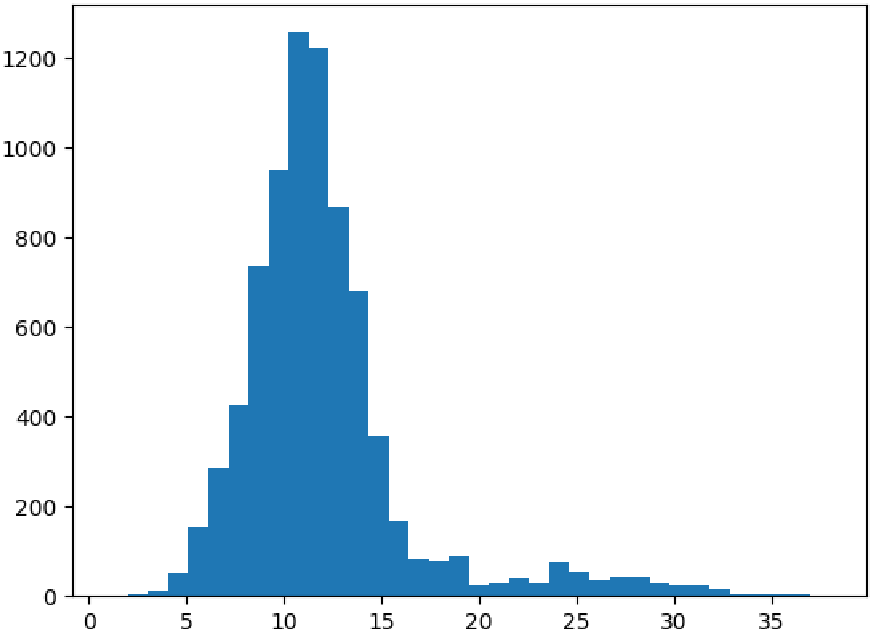
```
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[ ]:

```
20594
```

In [ ]:

```
lengths = [len(t.split(' ')) for t in tweets]
plt.hist(lengths, bins=len(set(lengths)))
plt.show()
```



based on the graph, we'll choose 35 as the maximum number of words per tweet

In [ ]:

```
maxlen = 35
def get_seqeuences(tokenizer, tweets):
  sequences = tokenizer.texts_to_sequences(tweets)
  padded = pad_sequences(sequences, truncating="post", padding="post", maxlen=maxlen)
  return padded
```

In [ ]:

```
padded_tweets = get_seqeuences(tokenizer, tweets)
padded_tweets_df = pd.DataFrame(padded_tweets)
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(padded_tweets_df, np.array(labels), test_size=0.15, random_
```

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
```

In [ ]:

```
model = Sequential()
model.add(Embedding(vocab_size, 20, input_length = maxlen))
model.add(Bidirectional(LSTM(20, return_sequences=True)))
model.add(Bidirectional(LSTM(20)))
model.add(Dense(1, activation = "sigmoid", kernel_regularizer='l1_l2'))
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_7 (Embedding)     (None, 35, 20)            411880    
                                                                 
 bidirectional_14 (Bidirecti  (None, 35, 40)           6560      
 onal)                                                           
                                                                 
 bidirectional_15 (Bidirecti  (None, 40)               9760      
 onal)                                                           
                                                                 
 dense_7 (Dense)             (None, 1)                 41        
                                                                 
=================================================================
Total params: 428,241
Trainable params: 428,241
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0005), metrics=['accuracy'])
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=128, callbacks=[EarlyStopp
```

```
Epoch 1/10
53/53 [==============================] - 17s 195ms/step - loss: 0.7057 - accuracy: 0.7436 - val_loss: 0.6587 - v
al_accuracy: 0.7332
Epoch 2/10
53/53 [==============================] - 5s 92ms/step - loss: 0.5620 - accuracy: 0.7571 - val_loss: 0.4089 - val
_accuracy: 0.8712
Epoch 3/10
53/53 [==============================] - 5s 102ms/step - loss: 0.3317 - accuracy: 0.9006 - val_loss: 0.3623 - va
l_accuracy: 0.8830
Epoch 4/10
53/53 [==============================] - 3s 51ms/step - loss: 0.2653 - accuracy: 0.9312 - val_loss: 0.3808 - val
_accuracy: 0.8906
Epoch 5/10
53/53 [==============================] - 3s 49ms/step - loss: 0.2345 - accuracy: 0.9441 - val_loss: 0.3726 - val
_accuracy: 0.8763
Epoch 6/10
53/53 [==============================] - 2s 43ms/step - loss: 0.1999 - accuracy: 0.9577 - val_loss: 0.3927 - val
_accuracy: 0.8704
Epoch 7/10
53/53 [==============================] - 3s 50ms/step - loss: 0.1960 - accuracy: 0.9557 - val_loss: 0.3721 - val
_accuracy: 0.8830
Epoch 8/10
53/53 [==============================] - 2s 45ms/step - loss: 0.1614 - accuracy: 0.9691 - val_loss: 0.3791 - val
_accuracy: 0.8712
```

Out[ ]:

```
<keras.callbacks.History at 0x7f6dcc8e9870>
```

In [ ]:

```
DL_model_accuracy = model.evaluate(x_test, y_test)[1]
DL_model_accuracy
```

```
38/38 [==============================] - 0s 6ms/step - loss: 0.3808 - accuracy: 0.8906
```

Out[ ]:

```
0.8905723690986633
```

Conclusion

In [ ]:

```
predictions = ((y_pred_bow_log, y_pred_bow_dt, y_pred_bow_gnb, None),
           (y_pred_tfidf_log, y_pred_tfidf_dt, y_pred_tfidf_gnb, None),
           (y_pred_sg_log, y_pred_sg_dt, y_pred_sg_gnb, None),
           (y_pred_cbow_log, y_pred_cbow_dt, y_pred_cbow_gnb, None),
           (None, None, None, DL_model_accuracy))
```

In [ ]:

```
rounded_accuracy_scores = []
for item in predictions:
    temp = []
    for val in item:
        if isinstance(val, np.ndarray):
            temp.append(round(accuracy_score(y_test, val) * 100, 2))
        elif isinstance(val, float):
            temp.append(round(val * 100, 2))
        else:
            temp.append(None)
    rounded_accuracy_scores.append(temp)
```

In [ ]:

```
algorithms = ("Logistic Regression", "Decision Tree", "Naive Bayes", "Deep Learning")
models = ("Bag of Words", "TFIDF", "Skipgram", "Cbow", "Deep Learning")

results_df = pd.DataFrame(rounded_accuracy_scores, columns=algorithms)
```

```
results_df['models'] = models
results_df.insert(0, 'models', results_df.pop("models"))
results_df.set_index('models', inplace=True)
```

Accuracy scores dataframe

In [ ]:

```
results_df
```

Out[ ]:

	Logistic Regression	Decision Tree	Naive Bayes	Deep Learning
models				
Bag of Words	88.05	84.85	78.96	NaN
TFIDF	88.05	83.75	79.04	NaN
Skipgram	88.30	83.67	82.74	NaN
Cbow	84.60	80.47	77.27	NaN
Deep Learning	NaN	NaN	NaN	89.06

We can conclude the Deep Learning model gives the best accuracy score of 89.06%